Ultracomputer Note # 28

"NETSIM" NETWORK SIMULATOR FOR THE ULTRACOMPUTER

Marc Snir

May 1981

1.0 INTRODUCTION

The ultracomputer project at NYU has as its target the design, analysis and construction of a shared-memory MIMD parallel computer. In such a system a large number of processors are sharing a common memory and each processor is executing its own code independently . The WASHCLOTH system simulates such an architecture, with each processor executing an expanded CDC instruction set [UCN12, UCN21]. This simulator assumes that access to memory, whether private or shared, is done in one instruction cycle. In a real machine, requests to shared memory would be processed through a communication network, and require more time to be satisfied.

This note describes an extension to WASHCLOTH that introduces extra delays for references to shared variables. The simulator is not intended to yield accurate measurements of the performance of an actual machine. This would depend on many factors which are not yet known, such as the precise

configuration (cache, local memory, etc.), the precise memory management system used, and the performance parameters of the hardware. Moreover, a precise simulation of the communication network we intend to implement would be extremely time and space consuming. We made several simplifying assumptions, and used some rough estimates on the relative performance of different hardware components of the system. Also, rather than tracing accurately each memory request through the network, we assume that memory requests are randomly distributed among memory modules, and perform a Monte Carlo simulation, based on the measured load on the network. This is expected to yield measurements similar to those one would obtain on a real network for randomly distributed memory references.

The reader is referred to [UCN25] for a brief description of the architecture we consider, and to [KS] for a detailed analysis of the performance of the communication network. We present in the next section the probabilistic model used. In section three we detail the information used to drive the simulation, and then discuss the different simplifying assumption made. A summary of results obtained follows, and we end with a discussion of these, and suggestions for further work.

2.0 NETWORK MODEL.

The communication network we are planning to use is a buffered Omega-network, which functions as a packet switching network. n processors are connected to n memory modules through (lg n)/2 stages of 4x4 switches. Each request to memory is issued as a set of packets with an address and data. The packets are forwarded through the successive stages of the network. The fundamental property of the Omega-network is that there exist a unique path from each processor to each memory module, and that the routing of a message at a switch at stage k depends only on the k-th bit of the destination address. For each output the switch keeps a queue of messages waiting to leave through this output. Each switch is able to handle in one "cycle" one incoming message on each input line and one outgoing message on each output line. The messages coming back from memory are handled in the same manner, with the same network used in the reverse direction.

This structure exhibits the following properties:

1. The minimal (one way) transit time of a message through the network is .5lg n cycles, even in absence of any load on the network.

2. As the network becomes loaded, queueing delays increase, and adversely affect the transit time.

3. Each queue in the network has a finite capacity, and will not accept new messages when it is full.

NETSIM models a system that exhibits the same global properties, but is much easier to simulate.

Consider the following problem: Assume that at each stage we have n queues, and that each queue may contain up to c messages. Assume that the number of messages at two subsequent stages is m1 and m2 respectively. What is the number of messages expected to move from the first stage to the second in one cycle?

A crude approximation to the correct answer may be obtained by using the following model:

Assume that the distribution of messages over the n queues corresponds to a random partition of a set with m1 (m2) elements into n subsets, each containing at most c elements. The expected number of nonempty queues at the first stage is (for large c) approximately equal to $n(1-(1-1/n)^{m1})$. The expected numbers of nonfull queues at the second stage is approximately equal to

$n(1-(1-1/n)^{(cn-m2)})$. Thus, the expected number of messages transmitted is approximately
$n(1-(1-1/n)^{m1})(1-(1-1/n)^{(cn-m2)})$.

We define a queueing system where the expected number of messages moving from stage to stage is given by the above formula. The system consists of p stages, one for each stage of the original network. Let m[s], s = 1,...,p, be the number of messages at stage s at the beginning of a cycle. For each message an independent random decision is made whether the message is moved to the next stage during this cycle: A message at stage s is moved to the next stage with probability
$(1-(1-1/n)^{m[s]})(1-(1-1/n)^{(cn-m[s+1])})n/m[s]$.

This system has the same qualitative properties as the original network, and hopefully, the same global behaviour. The behaviour of individual messages is not respected, however.

Our simulation embodies the following assumptions on the machine architecture:

1. The machine consists of 4K processors, and 4K memory modules. The communication network (which is built of 4x4 switches) consists of six stages in each direction. Thus, a message to the memory and back has a minimum transit time of fourteen cycles, assuming that two cycles are needed for the memory reference. In addition, each message is queued twelve times.

2. Each queue has a maximal capacity of 15 packets (This number is quite arbitrary. We have not however found much difference when it was reduced to five.)

3. Because of bandwidth limitations, requests are forwarded in separated, consecutive packets. A packet contains either an address and instruction code or half of a memory word. Therefore, each load command is transmitted as one packet but returns as three; A store command leaves as three packets and returns as one; A replace-add command always consists of three packets. Each request is therefore delayed at least by two more cycles.

4. The network cycles time (the time required for a message to move from one stage to the next one) is half of the time needed to execute one instruction.

We have simulated a system consisting of sixteen stages. During each instruction cycle two network cycles are simulated. Messages are accepted to the first stage, and moved from stage to stage probabilistically, according to the probabilities previously given (where m[0] = inf. and m[p+1] = -inf.). Although we generate only one message per command, when counting the number of messages at each stage, each message is weighted according to the number of packets it consists of. A message at the last stage leaves the network in one cycle (no queueing delays).

The minimal delay of a message through the network is eight instruction cycles (sixteen network cycles). The delay increases as the network is loaded. The network bandwidth is bounded by one packet per processor per cycle: Let l be the average number of loads, s be the average number of stores and r be the average number of replace-add's executed per processor, per cycle. Then the restricted network bandwidth implies that $l + 3s + 3r < 1$ and $3l + s + 3r < 1$ (the first inequality is obtained by

weighting each request by the number of packets needed to transmit it to memory, the second inequality is obtained by weighting each request by the number of packets used to transmit it from memory).

3.0 IMPLEMENTATION

We now describe NETSIM, the WASHCLOTH extension that implements the model described in the previous section. The reader is referred to [UCN12] and [UCN21] for a description of the WASHCLOTH simulator.

The NETSIM simulator consists of a slightly modified version of WASHCLOTH81, with the addition of the four following routines:

1. Initialization routine (NETINIT)

2. Memory reference trap (NETREF)

3. Network simulation (NETMOVE)

4. Summary (NETSUM)

In addition to WASHCLOTH storage this system uses one network request buffer for each processor, and a global network buffer.

3.1 Memory Reference Trap

A call to this routine is issued by WASHCLOTH whenever it simulates a memory reference instruction (load, store, replace-add).

If the memory location referred is global, and the network request buffer of the processor is empty, a message is entered into the buffer. The message contains a description of the request (instruction type, processor and register number, address, cycle number). If the network request buffer is full the request is not accepted, and the instruction is simulated anew at the next cycle. Otherwise the instruction is executed. However, if the instruction is a load or replace-add, the data is not available until the message traverses the network: A flag is set to indicate that the affected X register is not available. This flag will be cleared when the corresponding message comes out of the last network stage. Any subsequent instruction that tries to use the value of this X register while the flag is set will not be executed but instead will be retried next

cycle. Also, an attempt to issue a load (or replace-add) using an X register whose flag is set results in an error.

3.2 Network Simulation.

At the end of each instruction cycle two network cycles are simulated. First the transition probabilities are computed. Next, for each message in the global network buffer a probabilistic decision is made whether to augment its stage counter by one, and for each message in a network request buffer, a probabilistic decision is made whether to move the message into the global network buffer. This routines also clears the flags described above whenever a request leaves the network.

3.3 Summary

The summary routine NETSUM can be called at any stage of the simulation to obtain cumulative statistics concerning the number of instruction cycles simulated and the data memory requests trapped.

To summarize: Network traffic is generated for each data reference to global memory, according to the model previously presented. An instruction that uses data from

global memory is delayed until that data is loaded. A reference to global memory may be delayed if the network request buffer is full.

4.0 DISCUSSION

Since our simulation is not "faithful" doubts on the relevancy of results obtained from this simulation are legitimate. We would like to pinpoint some of the aspects of this issue.

1. Our probabilistic model is not faithful. As an experiment we ran on NETSIM synthetic loads with a random, steady-state distribution of replace-add requests, and compared the results with those obtained from a "faithful" simulation of one network stage under the same distribution. The results are displayed in table 1.

Table 1

Average number of instr. cycles per request	4	5	6	7	8	9
Network transit time with faithful simulation	36	21	16	14	13	12
Network transit time with NETSIM	36	25	20	18	18	16

As we see, our model is consistently pessimistic. The reason is obvious: In a unloaded network the average delay per request at each switch will be nearly one, even if the request consists of three packets; The average delay in our model for a request consisting of three packets will be always larger than 1.5.

This discrepancy is smaller for a loaded network, and nonexistent for one packet messages.

There are at least two more possible sources of discrepancies in our model:

1. The actual distribution of requests varies in time. We do not know which model is more sensitive to fluctuations in arrival rates.

2. The actual requests are not randomly distributed over the processors and memory modules. We expect to achieve a near random distribution by a judicious use of hardware (address hashing) and software (even loading of processors) mechanisms.

2. The code run is not compiler optimized to reduce delays due to global memory accesses: There is no prefetching of operands or reordering of operations. These could improve the performance significantly.

3. The machine is assumed to contain 4K processors, but actually only a small number of processors are simulated. The load on the network is obtained by assuming that the activity of the processors simulated is a representative sample of the overall

activity in the machine. This assumption increases the variance of that activity.

4. We do not take into account the network traffic created by instruction fetches. It is hopped a suitable cache mechanism will prevent most of it.

5. We do not take into account merging in the network, which can only improve performance.

6. We do not take into account the savings in global memory references to be expected from a judicious use of a data cache or a local memory.

7. We do not take into account the network traffic and delays due to references to private variables. It is expected that private variables will be kept mainly in local memory, or cached, with little global traffic generated.

8. We do not model overheads introduced by the operating system: loading and unloading tasks, I/O, etc. There is no excuse for this. On the other hand the operating system will also alleviate some of the problems, especially if each processors will be multiprogrammed: busy waiting may be

replaced by interrupt mechanism.

5.0 SUMMARY OF RESULTS

We have run the simulator on several scientific codes:

1. A parallel version of a part of NASA's weather code (solving a two dimensional PDE), with 16 processors.

2. The same program, with 48 processors.

3. A tridiagonal symmetric linear system solver, with 16 processors.

4. A multigrid Poisson PDE solver, with 16 processors.

The results of the simulations are summarized in the tables below. The transit time for requests through the network is measured in instruction cycles.

Table 2

problem number	instruction cycles without delay	instruction cycles with delay	local memory references	global memory references	average transit time
1	8078	12949	26855	16680	8.94
2	3828	5766	30715	22941	8.83
3	45217	57979	184018	43348	8.81
4	256590	319318	900303	323146	8.85

Table 3

problem number	extra delay	ratio instruction to memory references	ratio instruction to global memory references	ratio local to global memory references
1	+60%	4.8	12.4	1.6
2	+63%	5.2	12.1	1.3
3	+28%	4.1	21.4	4.2
4	+24%	4.2	15.8	2.8

Table 4

problem number	extra instructions with delay	loads and replace-adds	ratio
1	77936	14702	5.3
2	93024	20819	4.5
3	204192	41528	4.9
4	1003648	288658	3.5

We wish to make several remarks concerning these

numbers.

1. We did not present statistics on the number of global memory requests delayed because of a full network request buffer. This number never exceeded a few percents.

2. The numbers in column three of table 3 indicate that only 20% to 25% of the instructions executed generate memory references. This is due both to the peculiarities of the CDC machine language and to the fact that all the codes are computation bound.

3. The higher delays registered for the first two problems are clearly due to the larger number of global memory references issued. The weather code is a slight modification of a serial code, and no particular effort was made to minimize the number of global memory references [UCN22]. In the two other codes an attempt was made to reduce the use of global variables by judicious copying into local variables.

4. The average transit time of requests through the network was very close to the minimum (eight instruction cycles). The reason is that the load on the network was far beneath its maximal capacity (one packet per network cycle per processor, which is more than one request every 1.5 instruction cycle).

5. The ratio of the number of "idle instructions" (extra instructions executed when global requests are delayed) to global loads is slightly more than half the average network delay for the first three programs. The low value obtained for the last program is an artifact of this program: A substantial part of the instructions executed are loads implementing a busy wait. The number of global loads goes down by 30% when global requests are delayed, artificially improving the performance of this program under NETSIM. We believe the first three programs to be more representative of real performance and we expect that on the average a program will be delayed for four to five extra instruction cycles for each load to

global memory. We expect this ratio to be nearly independent of the particular application programmed, and to depend more on the machine language and the compiler used. If so, the ratio can be used to predict the running time of programs when delayed given their running time when not delayed and the number of global memory requests they generate.

6. The overhead in CPU running time for our simulator over the WASHCLOTH simulator ran from a factor of 10 to a factor of 5, with overhead decreasing as the number of processors is increased.

We tried to change the parameters of the model, to check how sensitive are our results to the particular assumptions embodied in our model. The performance was very slightly affected when we decreased the buffer size from fifteen to five. This was to be expected, as in all our examples the load is very light.

A more drastic effect was obtained by slowing down the network by a factor of two. The extra delay increased for the first and third problem to 128% and 61% respectively. The average transit time through the network became 17.6 and 17.2 respectively and the ratio between the number of idle instructions and the number of loads and replace-adds went to 11.2 and 11.4 respectively.

We also tested the result of doubling the number of packets needed for each request (that is decreasing the bandwidth of each line in the network by a factor of two). The extra delay increased to 68% and 31%; The average transit time became 10 and 9.7, and the ratios of idle instructions to global loads were 6.0 and 5.9 respectively. Thus, halving the capacity of the network does not seem to have a major impact, for the programs in our sample. This is obviously due to the fact that we operated far away from the maximal capacity.

6.0 FURTHER RESEARCH

We intend to continue testing new programs, and diversify our statistics. A more realistic queueing model will be used. Also, a faithful simulator for the network will be completed soon. Although this simulator will be of course much slower, we hope to use it in order to tune the parameters of the current simulator and obtain more accurate statistics. With the expected upgrading of WASHCLOTH to include a primitive operating system, our simulator will be upgraded as well. The subsequent versions of NETSIM will reflect some of the factors currently ignored, such as instruction fetches, and will be used to test for the efficiency of various memory management policies.

7.0 REFERENCES

[KS] Clyde Kruskal and Marc Snir, "Analysis of Omega-Type Networks for parallel Processing", in preparation.

[UCN12] Allan Gottlieb, "WASHCLOTH - The Logical Successor to SOAPSUDS." Ultracomputer Note #12, Oct. 1980.

[UCN21] Allan Gottlieb, "WASHCLOTH81". Ultracomputer Note #21, Jan. 1981.

[UCN22] Norman Rushfield, "Atmospheric Computations on Highly Parallel MIMD Computers". Ultracomputer Note #22, Feb. 1981.

[UCN25] Allan Gottlieb and J. T. Schwartz, "Networks and Algorithms for Very Large Scale Parallel Computations". Ultracomputer Note #25, March. 1981.